

---

## CEPL (Code Evaluate Play Loop) - [Beta]

CEPL is a lispy and REPL-friendly Common Lisp library for working with OpenGL.

Its definition of success is making the user feel that GPU programming has always been part of the languages standard.

The usual approach to using CEPL is to start it at the beginning of your Lisp session and leave it open for the duration of your work. You can then treat the window it creates as just another output for your graphics, analogous to how `*standard-output*` is treated for text.

CEPL is in beta. The API is close to what it needs to be but there are still many bugs to fix, features to add, and experiences to smooth out.

See the [cepl.examples](#) repository for some examples of how CEPL can be used

Videos: [http://www.youtube.com/playlist?list=PL2VAYZE\\_4wRKKr5pJzfYD1w4tKCXARs5y](http://www.youtube.com/playlist?list=PL2VAYZE_4wRKKr5pJzfYD1w4tKCXARs5y)

### Installing

Run `(ql:quickload :cepl)` at your REPL.

### Cloning

Whilst it is recommended to get CEPL from quicklisp, if you clone please note that `master` is not the stable branch. Please use `release-quicklisp` for the stable code that will be in the next CEPL release.

### Documentation

Currently we have full documentation of every exported symbol in the CEPL package. You can find this here: [CEPL API Docs](#)

Guides will be provided in future, however these take much longer to write.

I can also be reached by my email (techsnuffle [at] gmail · com) and sometimes on #lispgames IRC. Come to #lispgames anyway though, there are some lovely folks, all lispy dialects welcome!

### Requirements

All of the following will be downloaded automatically by quicklisp

- 
- `cl-ffi`
  - `cl-opengl`
  - `cl-plus-c`
  - `cl-ppcre`
  - `documentation-utils`
  - `ieee-floats`
  - `float-features`
  - `named-readtables`
  - `varjo`
  - `bordeaux-threads`

**C Library dependency** CEPL uses OpenGL ( version  $\geq 3.1$  ) so you need to make sure it is available on your machine. Installing your GPU drivers will usually handle this.

**CEPL's Host** CEPL abstracts working with OpenGL but is not responsible for creating a window or GL context; this is handled by a [Host](#). Right now the only supported host is [SDL2](#); the system to load is called `cepl.sdl2`, you can find it here: `cepl.sdl2`

## Getting Started

*Note:* On [macOS](#), [slime](#) users will need to read [docs/single-thread-swank.md](#) to deal with a complication specific to projects interacting with the window manager. You can then follow the rest of this guide as usual.

To load CEPL and the default host (`sdl2`) do the following:

- `(ql:quickload :cepl.sdl2)`
- `(cepl:repl)`

You should see an empty window appear, OpenGL is now initialized, and you can use CEPL as you like.

## Making a CEPL Project

The best way to get started is to make a new project that uses CEPL. Do the following in your REPL to get set up:

- First, run `(ql:quickload :cepl)`

- 
- Then run `(ql:quickload :quickproject)`. CEPL uses this to create a lisp project using its own templates
  - Then run `(cepl:make-project "my-proj")`. This will use quickproject to make a new project with all the correct dependencies. Remember that cepl does not handle window managers or input so by default your new project will use the following
    - cepl for the graphics
    - cepl.sdl2 for the host
    - skitter for handling input and events
    - dirt for loading images There are many good event systems & image loaders out there so browse around. But here are two :)

You are now ready to get started. Simply run: - `(ql:quickload "my-proj")` - `(in-package :my-proj)` - and finally (if you havent already) `(cepl:repl)`

**Windows C Library Hack** If you are having issues getting the C libraries to load and just need to rule out whether Lisp can find them, try putting them in the same folder as the lisp exe. For example `C:\Program Files\sbcl\`.

## CHANGELOG

This mainly covers features & news rather than individual bugfixes. When we are out of beta these will be covered more often

### 2019-05-05

- Added # `'clear-attachments`
- `clear-fbo` now let's you specify attachments to clear
- Added `with-outputs-to-attachments` which lets you rebind which outputs from your pipelines map to which of the currently bound fbo attachments
- Deprecated `attachment-pattern` in favor of `color-attachments`
- Can now specify a discarded attachment in `color-attachments` with `+discard-attachment+`

### 2019-04-XX

- stability work on `with-context-state-restored`. More coming to this soon.

---

## 2019-02-XX

- Add #'default-fbo accessor

## 2019-01-20

- Return from work crunch to add anisotropy support to samplers :)

## 2018-09-29

- Added support for MultiDrawIndirect using `multi-draw-g`
- Added support for base-vertex in buffer-streams

## 2018-08-10

- `wait-on-gpu-fence` now expects an (`unsigned-byte 64`) or `nil`. `nil` is used to indicate no timeout and anything else is the timeout in nanoseconds
- added `copy-g` as an more general alternative to `push-g` & `pull-g` for moving data.
- exported the many typed functions `copy-g`, `push-g` & `pull-g` use behind the scenes for transfers
- fixed serious bug in `clear` which, when passed no fbo, would try to clear both the read & draw fbo attachments. Now only clears draw as per the gl spec.
- `make-texture` can now take `gpu-arrays` as the 'initial-contents' argument.
- transfers between buffer & texture backed gpu arrays now work
- c-arrays & gpu-arrays now accept a row-alignment value on creation. This defaults to 1 but can be 1, 2, 4 or 8
- texture uploads/downloads now correctly take row alignment into account.
- Fixes to ensure mutable textures are 'complete' on creation
- fix issues when creating c-arrays from lisp arrays that stemmed from specification of dimensions being (x-size y-size) in CEPL/GL and (column-size row-size) in lisp.

## 2018-07-15

- added `make-gpu-arrays-from-buffer-id` which lets you specify a gl buffer-id and layouts of the data and get a CEPL gpu-buffer in return
- added `make-gpu-array-from-buffer-id` which is shorthand for the above when you only need one gpu-array.

- 
- added option to `make-gpu-buffer-from-id` so you can pass layouts instead of initial-contents. This is handy when you allocate and layout out the buffer storage without providing new data.
  - add a `keep-data` argument, to a bunch of buffer related functions which take layouts. This lets you create the buffer object, specifying the content layout without replacing the data.
  - added `make-buffer-stream-from-id-and-layouts`. Makes it possible to make streams out of vaos without having gpu-arrays.
  - print-object for `gpu-buffers`
  - misc bug fixes

**2018-04-02**

Headlines:

The most important change this release is that lambda pipelines now recompile when the functions they use as stages are recompiled. This change comes with a performance cost of course so if you wish to opt out of this (recommended for when you ship your project) then pass `:static` in the context argument to `pipeline-g`

Also if you are using GL $\geq$ 4.3 you can now create empty fbos, which are handy if you want to use the fragment stage without writing data to bound textures (compute-type fun to be had here)

Other important changes:

- `viewport-resolution-x` & `viewport-resolution-y` had been documented as returning `single-floats` but were returning unsigned ints, this now fixed.
- I'm an idiot that didn't know about `~/swank.lisp`, please see the new 'single-thread-swank' advice above if running on macOS.

**2018-02-17**

## **BREAKING CHANGES**

Due to the changes listed below you will now want to change your current asd's to require `:rtg-math.vari` and to change your packages to `:use :rtg-math`.

Some folks were asking for the ability to use their own math libraries rather than `rtg-math` which CEPL has depended on up until now. I have made this change but this means that this is something users will have to add themselves from now on.

For those picking a library cepl expects vectors & matrices to be sized simple-arrays of single-floats e.g. `(make-array 3 :element-type 'single-float)` for 3 component vectors.

---

We have also removed the dependency on fn & cl-fad.

Other Changes: - added the remove-surface function - closing primary window no longer quits cepl - removed interactive from lifecycle events (will add surface lifecycle events in future release)

## **2018-01-11**

\*I am terrible at changelogs. I am retroactively making the updates for everything between june 2017 and jan 2018

- Added support for using gpu-lambdas inline in `defpipeline-g` forms
- Lots of work on the `cepl.tests` project resulting in fixes for:
  - gpu-lambdas
  - lambda-pipelines
  - struct uniforms
  - uniform baking

## **2017-12-??**

- Add render-buffer support
- Fix struct layout bugs. SSBOS work correctly now
- Add funcall-g (limited support for calling gpu-functions from cpu directly)
- Remove the deprecated `g->`, `def-g->` & `glambda` macros

## **2017-11-??**

- Add compute support
- Add SSBO support (though currently hobbled by struct layout bug)
- Add single stage pipeline support
- Add transform feedback support
- Add shared context support
- Add gpu-fence support
- Add gl query support
- Complete rewrite of pipeline state cache (now faster and safer)
- Avoid recompilation of pipelines where dependencies have not changed
- Rewrite struct code to get away from some dependencies

---

## **2017-10-??**

- Multiple (non-shared) context support

## **2017-09-??**

- Add border-color support

## **2017-08-??**

- Mostly bugfixes

## **2017-07-??**

- Add instance array support (per instance data in pipelines)
- Add stencil support
- Add scissor support
- Add color-mask support
- Add multisample support
- renaming macros with confusing/inconsistent naming
- blending param caching on context (not optimized yet)
- Make #free work with pipelines

## **2017-06-??**

- You can now modify a buffer-stream's primitive type after creation
- experimental profiler
- Huge improvements to per-frame performance through:
  - typing of functions & inlining where it made sense
  - fixing bug where uniforms ids queried too often
  - macros to reduce frequency of fetching the context object
  - much better state caching in context object
  - avoiding cffi enum conversion when lookup functions are faster
  - precompute & cache more size info in wrapper types

- 
- attachments have their own viewports, no more computing of viewport during draw
  - dont set viewport on every map-g (this was unnecessary)
  - remove some of the 'with-\*' macros whos contract forced us to do more state changes/caching that we would otherwise like.
  - pipelines can be declared static. This tells cepl to type the generated functions.
  - a huge pile of other small changes

#### 2017-06-04

- pipelines can take `:dynamic` as their draw mode. This means they will take the draw-mode from the `buffer-stream` they are mapped over. This only works for pipelines with `vertex` & `fragment` stages.
- `buffer-streams` now hold the primitive type of their data. Defaults to `:triangles`
- Fix bug that was stopping g-structs contain matrices
- Cache more values in `buffer-stream` to save time during rendering
- Add `surface-dimensions`, `surface-resolution`, `surface-title` & `surface-fullscreen-p`
- add `adjust-gpu-array` (pretty basic right now)
- Remove `cepl.spaces`, It is now a seperate project (and will be in quicklisp in the next cycle)
- Remove `cepl.misc`. If you were using the `draw-texture` function then please consider `Nineveh` (which will be in quicklisp in the next cycle)
- `make-project` now uses `dirt` instead of `devil`. `dirt` uses `cl-soil` which ships with binaries for multiple paltforms so has a better 'out of the box' experience (plus also supports more formats)

#### 2017-05-16

*I missed some logs here so this is a recap of everything since 2017-02-19*

- Geometry & Tessellation fully supported (including for inline glsl stages)
- Draw mode can now be specified for pipelines
- fixes for pull-g with gpu-functions & pipelines



- 
- add `with-gpu-array-range-as-pointer` & `with-gpu-array-range-as-c-array`. These still feel experimental to me.
  - add `reallocate-gpu-array` & `reallocate-buffer`
  - buffer-streams always hold on to their gpu-arrays by default
  - Refactoring based on changes in Varjo
  - Added bordeaux-threads as a dependency. Will be needed for some context related things
  - Very basic support for multiple surfaces (windows)
  - New 'host' api. Is versioned so old hosts are still supported
  - remove `run-session`. All of these attempts at thread hackery felt bad. I'm sticking with `slime-style` until we have a better fix

#### **2017-02-19**

- Removed the `continuable` macro. The macro can be found in the `livesupport` project. Simply `(ql:quickload :livesupport)`